Kristin Wright
University of Utah
Department of Computer Science
50 S. Central Campus Drive, Rm 3190
Salt Lake City, UT 84112-9205
kwright@CS.Utah.EDU
Voice: 1-801-581-4280
Fax: 1-801-585-3743

Symposium: Global Internet: Application and Technology

Using Reliable Multicast for Caching and Collaboration within the World Wide Web

The World Wide Web has become an important medium for information dissemination. One model for synchronized information dissemination within the Web is *webcasting* in which data are simultaneously distributed to multiple destinations. The Web's traditional unicast client/server communication model suffers when applied to webcasting; approaches that require many clients to simultaneously fetch data from the origin server will likely cause server and link overload.

In this paper we describe a webcast design that improves upon previous designs by leveraging application level framing (ALF) design methodology. We build upon the Scalable Reliable Multicast (SRM) framework, which is based upon ALF, to create a custom protocol to meet webcast's scalability needs. We employ the protocol in an architecture consisting of two reusable components: a webcache component and a browser control component. We have implemented our design using a new SRM library called *libsrm*. We present the results of a simple performance evaluation and report on lessons learned while using *libsrm*.

Using Reliable Multicast for Caching and Collaboration within the World Wide Web

Kristin Wright
University of Utah
kwright@cs.utah.edu

Steven McCanne
University of California, Berkeley
mccanne@cs.berkeley.edu

Jay Lepreau
University of Utah
lepreau@cs.utah.edu

Abstract

The World Wide Web has become an important medium for information dissemination. One model for synchronized information dissemination within the Web is *webcasting* in which data are simultaneously distributed to multiple destinations. The Web's traditional unicast client/server communication model suffers, however, when applied to webcasting; approaches that require many clients to simultaneously fetch data from the origin server using the client/server model will likely cause server and link overload.

In this paper we describe a webcast design that improves upon previous designs by leveraging application level framing (ALF) design methodology. We build upon the Scalable Reliable Multicast (SRM) framework, which is based upon ALF, to create a custom protocol to meet webcast's scalability needs. We employ the protocol in an architecture consisting of two reusable components: a webcache component and a browser control component. We have implemented our design using a new SRM library called *libsrm*. We present the results of a simple performance evaluation and report on lessons learned while using *libsrm*.

1 Introduction

Online collaborative environments designed to facilitate long-distance collaboration are enjoying growing popularity. Video, audio, and whiteboard collaborative applications have been widely deployed. To complement these applications, it would be useful to have the ability to distribute documents in a synchronized fashion over the World Wide Web. For example, a speaker might display slides directly on remote listeners' desktop web browsers. There are a number of slightly differing models, each requiring the synchronized distribution of web documents and their embedded objects to multiple sites. A common name for this type of collaborative session is *webcasting* [1, 3, 7, 9].

A simple webcast design might require each session *participant* to individually fetch the required document from the *origin server*, the web server on which the document resides. The problem with this simple solution is that the aggregate requests can lead to *implosion*, a pathology in which a resource is unable to keep up with an incoming stream of messages. Even if the server were able to keep up with the requests, the subsequent replies could lead to link overload.

To achieve scalability, neither the network traffic nor the number of server hits generated by a webcast session can be allowed to significantly increase with session membership. A more efficient approach than the simple design above would be to use IP multicast to achieve greater scalability [4]. One way to leverage multicast would be to modify the web server such that it multicasts data in response to requests from webcast program participants [1, 3]. While such an approach is more scalable than our naïve solution above, modifying the web server limits the usefulness of the resulting webcast application because server modification is only feasible when the server is within the local administrative domain.

A second multicast approach is to build a client-side web proxy to intercept participants' browser requests, fetch the data from the origin server, and multicast the data to session participants. [7] and [9] use this approach and further improve scalability by caching data locally so that browser requests for previously requested data need not be forwarded to the server. This approach capitalizes on multicast's scalability without requiring server modification but fails to prevent server implosion common in collaborative situations, e.g., when several students listening to a remote lecture simultaneously fetch a web document referenced by the lecturer.

Our webcast solution builds on the approach in [7] and [9], further addressing the server implosion problem by using the Scalable Reliable Multicast (SRM) framework [5] to build a custom protocol that addresses all of webcast's scalability needs. SRM embodies a design principle called Application Level Framing (ALF) [2] that applications can leverage to build custom protocols to suit their unique needs. The insight behind ALF is that applications know their requirements better than do generic communication protocols. Accordingly, SRM provides reliable multicast service while allowing the application to determine its own semantics such as packet ordering and framing.

Our contributions are that (i) we exploit SRM's

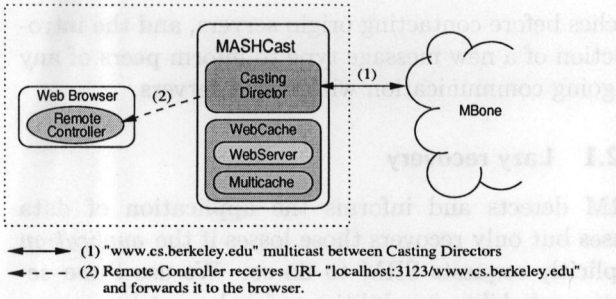

Figure 1: The URL `www.cs.berkeley.edu` is multicast among the Casting Directors. Upon receipt, the Casting Directors prepend the local Web Server's address onto the URL and forward the mangled URL to the Remote Controller.

flexibility to create a new protocol that maximizes webcast scalability by minimizing origin requests and network traffic, (ii) we embody our protocol in a novel webcast architecture and implementation called MASHCast that features several reusable components, is browser independent, and is compatible with the existing Web infrastructure, (iii) we present initial performance characteristics, and (iv) we report some lessons learned during implementation.

2 Dual component architecture

Our webcast architecture consists of a control component called the Casting Director and a multicast caching component called the Web Cache. The Web Cache consists of two subcomponents: a Web Server, which receives requests from the browser, and the Multicache, the multicast web page cache maintained using SRM.

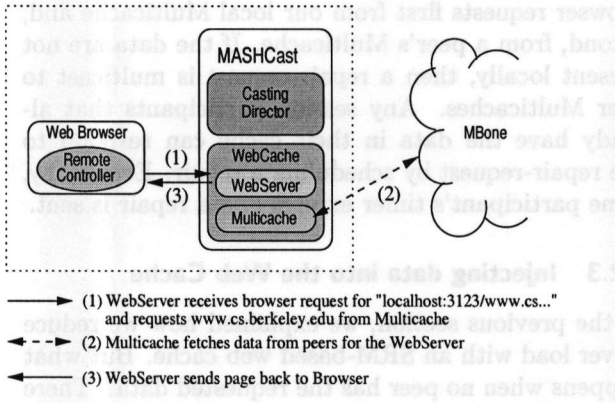

Figure 2: Since the requested data are not in the local Multicache, the Multicache requests the data from peer Multicaches.

To participate in a MASHCast session, the user need only start the MASHCast application and a web browser on the same machine and fetch a special MASHCast bootstrap page into the browser that loads a java applet we call the Remote Controller. The Casting Director and the Web Cache run in different processes on the same machine as the browser. They use separate multicast addresses to communicate with peer components in the same MASHCast session across the IP Multicast Backbone (MBone). The MASHCast application sits between the browser and the network.

Once MASHCast is started, a sender webcasts a page by specifying the page's URL to the Casting Director via a built-in GUI[1]. The Casting Director provides the synchronization in a webcast session by multicasting URLs to peer Casting Directors and forwarding received URLs to the browsers. Peer Casting Directors listen for URLs and, upon receipt, prepend the request with the URL of the local instance of MASHCast. The resulting *mangled URL* is forwarded to the Remote Controller. Figure 1 shows how the Casting Director forwards URLs to the browser and other Casting Directors. Because of our mangling scheme, there is no need to configure the browser's HTTP Proxy; MASHCast can coexist with any caching web proxy the user may have already configured and can also be configured to use such a proxy itself.

When the Remote Controller receives a URL from the Casting Director, it instructs the browser to fetch the corresponding document. Because the local Web Server's URL has been prepended to the original URL, the request is actually directed back to MASHCast. The Web Server subcomponent of the Web Cache receives the browser's page request and attempts to satisfy the request from the Multicache. If the local Multicache cannot satisfy the request from its own cache, it requests the data from peer Multicaches. If no Multicache in the session has the data, a single Multicache is selected in a distributed fashion to fetch the data from the origin server and then multicast the data to the group. Figure 2 shows the Web Cache satisfying the browser's request from the Multicache.

While we believe that our architecture's main contribution comes when used as a collaborative webcast application such as MASHCast, our architecture is novel because, in addition to being browser independent and compatible with the existing Web infrastructure, the components can be cleanly decoupled for reuse. The Multicache subcomponent can be used alone to cache not only Web pages and their embedded objects, but any type of data in a distributed, scalable, reliable manner. The Web Cache is not a suitable replacement for existing hierarchical web caching proxies, but it can be used independent of the Casting

[1] URLs can actually be passed to the Casting Directors using any mechanism that multicasts the URL to the Casting Directors' multicast address. The GUI is a convenience for the user, not a necessity of the design.

Director as a passive multicast web cache in groups where a caching proxy server is not already in place. In fact, we have already separately used most of the components. We have used the Web Server alone for recording browser requests, the Casting Director alone for an early demonstration, and the Web Cache alone for the experiments reported later.

3 Extending SRM

Web data are usually either HTML documents or images embedded within such documents. Browsers can faithfully render these types of data only if they are received in whole. Thus, our application's semantics require reliable delivery. In this section we describe SRM's reliable service framework and how we extend it to maximize our webcast protocol's scalability.

3.1 SRM's reliability mechanisms

A novelty of the SRM framework is its scalable multicast retransmission scheme which relies upon *repair-requests* for loss recovery. Group members are responsible for detecting loss and requesting retransmission. Repair-requests are satisfied with a *repair* message. Repair-requests and repairs are sent to the entire multicast group, or sent to a subset of the group if local recovery is in effect.

In order to prevent implosion of repair-request or repair packets sent from receivers to the group, SRM must suppress duplicate repair-requests and repairs. To this end, SRM employs a randomized and distributed damping mechanism. In this scheme, repair-requests and replies are sent only after a randomly chosen delay. The *suppression intervals* from which this delay is chosen are biased by the round-trip time to the node that triggered the request. All session participants "listen" to all requests and replies and backoff any duplicate requests and cancel any duplicate replies that they may have scheduled.

All participants cache old data and any participant with a copy of the data can send a repair message. SRM sources periodically send *session announcements* reporting their current state. Participants check their state against session announcements to detect tail losses.

3.2 SRM customizations

SRM's basic reliability mechanisms alone would meet webcast's *reliability* requirement. However, without specialization, the protocol would not *scale* as desired. In this section, we describe three customizations to the SRM framework that maximize scalability: an optional, custom recovery policy that ignores data deemed unimportant by the user, a check of peers' caches before contacting origin servers, and the introduction of a new message type to inform peers of any ongoing communication with origin servers.

3.2.1 Lazy recovery

SRM detects and informs the application of data losses but only recovers those losses if the *application* explicitly requests SRM to do so. We use these *selective reliability* capabilities to implement two recovery policies for MASHCast. Our *aggressive-recovery* policy specifies that lost data are always recovered; the *lazy-recovery* policy specifies that the SRM layer only recover data that the browser has explicitly requested. Using lazy-recovery can yield slower response times than those of aggressive-recovery due to the overhead of retrieving data from a peer's cache rather than from one's own cache. However, lazy-recovery can save cache space and reduce network traffic in many circumstances—for example, when late joiners in a collaborative session aren't interested in seeing previously webcast data or when the Web Cache is being used as a passive cache.

3.2.2 Leveraging peers' caches

When a URL is webcast, ideally only a single Multicache fetches the data from the origin server and multicasts it to the group. [7] and [9] designate the node where the request originated as the node that sends the origin request. Without additional changes in the protocol, however, this policy could fail to prevent server implosion caused by the synchronization inherent in common collaborative situations, e.g., when many students listening to a remote lecture simultaneously fetch a web document referenced by the lecturer.

To avoid server implosion, we attempt to satisfy browser requests first from our local Multicache and, second, from a peer's Multicache. If the data are not present locally, then a repair-request is multicast to peer Multicaches. Any session participants that already have the data in their cache can respond to the repair-request by scheduling a repair. Eventually, some participant's timer expires and a repair is sent.

3.2.3 Injecting data into the Web Cache

In the previous section, we explained how we reduce server load with an SRM-based web cache. But what happens when no peer has the requested data? There must be some mechanism to retrieve data from the origin server and inject it into the multicast cache.

One solution is to have each participant set a repair timer regardless of whether or not it has the data. Should a participant who does not have the data set a repair timer that expires before other repairs are received, that participant sends an origin request when

its timer expires rather than a repair. Although this solution works, it can lead to duplicate origin requests because, during the interim period when the data are being retrieved from the origin server, other participants' timers may have expired, thereby triggering more origin requests.

To solve this problem, we leverage SRM's flexibility by defining a new message called a *reply-pending* message which notifies other participants that an origin request has commenced, thereby suppressing duplicate origin requests. Upon receipt of a repair-request message, nodes that do not have the requested data set a reply-pending timer rather than a repair timer. If a node's reply-pending timer goes off, that node sends a reply-pending message to the group *and* fetches the data from the origin server. All session participants listen for reply-pending messages and refrain from sending duplicate origin requests by cancelling their outstanding reply-pending timers. To further reduce spurious origin requests sent when the data exist in a peer's cache, we schedule reply-pending timers such that they probabilistically expire after repair timers. Reliability is still guaranteed because repair-request timers remain set.

The techniques presented in Secs. 3.2.3 and 3.2.2 together have the novel effect of applying SRM's slotting and damping technique to origin server requests. In this manner, origin requests are minimized—something we believe is vital for scalability.

4 Implementation

To verify the efficacy of our design and provide a useful tool for online collaboration, we have implemented a prototype of the MASHCast components within the MASH [8] environment. Our prototype implementation is optimized for ease of implementation and policy change and not for performance. Nevertheless, a simple benchmark test measuring document fetch times showed that our protocol greatly reduced the number of origin requests while improving the response time over response times when not using the Web Cache—98% when using aggressive-recovery and slightly (3%) when using lazy-recovery[2].

In addition to the simple benchmark test, we ran a second experiment to gain intuition of the behavior of our protocol. In each experiment, five participants webcast fourteen documents at deterministic times within the session.[3] All experiments were run thirty times and the results averaged. In the first set of runs, the Multicache started empty. In the second set of runs, a single participant's Multicache was primed with the desired documents. We extended `tcpdump` [6] to understand SRM message types, and used it to count repair-requests and reply-pending messages.

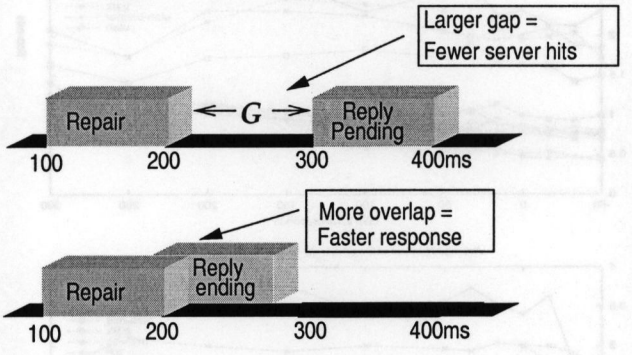

Figure 3: Timing tradeoffs.

The addition of reply-pending messages presents a new interval definition challenge. The challenge is to define the repair and reply-pending suppression intervals such that repair-requests are satisfied by the Multicache before the origin server, whenever possible. To gain some intuition about the relationship between these two intervals, we varied the reply-pending interval parameter across experiments. We define $\mathcal{G}$ to be the difference between the the right edge of the reply interval and left edge of the reply-pending interval, as shown in Figure 3. A negative $\mathcal{G}$ implies that we expect to generate a greater number of unnecessary origin requests when the data are already in the Multicache, but obtain shorter response times when the data are not in the Multicache; as $\mathcal{G}$ grows larger, we expect increasing response times when the data are not yet in the Multicache but decreased spurious origin requests when the data are cached.

Figure 4 shows our results. Most documents' response times behave as expected as $\mathcal{G}$ increases. The graphs on the left show the case where the Multicache is initially empty. The graphs on the right show the case where a single participant's Multicache has been primed. When the cache is initially empty, response times generally (clustered lines at bottom) linearly increase with $\mathcal{G}$ and the number of origin server requests hovers around one. On the right, when a single participant's Multicache is primed, the response times remain flat and the number of origin server requests converge to zero. `utah-mts`[4] (the top line) is an order of magnitude larger than the other documents and, therefore, takes longer to download and is responsible for more origin requests since participants will issue repair-requests (at increasing intervals) until the en-

[2]Experiment details omitted due to length can be found in our technical report [12].

[3]Although five nodes are not sufficient to test our protocol for scalability with respect to session size (future work), we do consider five nodes sufficient to test the protocol's general behavior.

[4]Document monikers were chosen to reflect origin servers' geographical location. Although not always true, in this case the geographical distance also reflects the network latency.

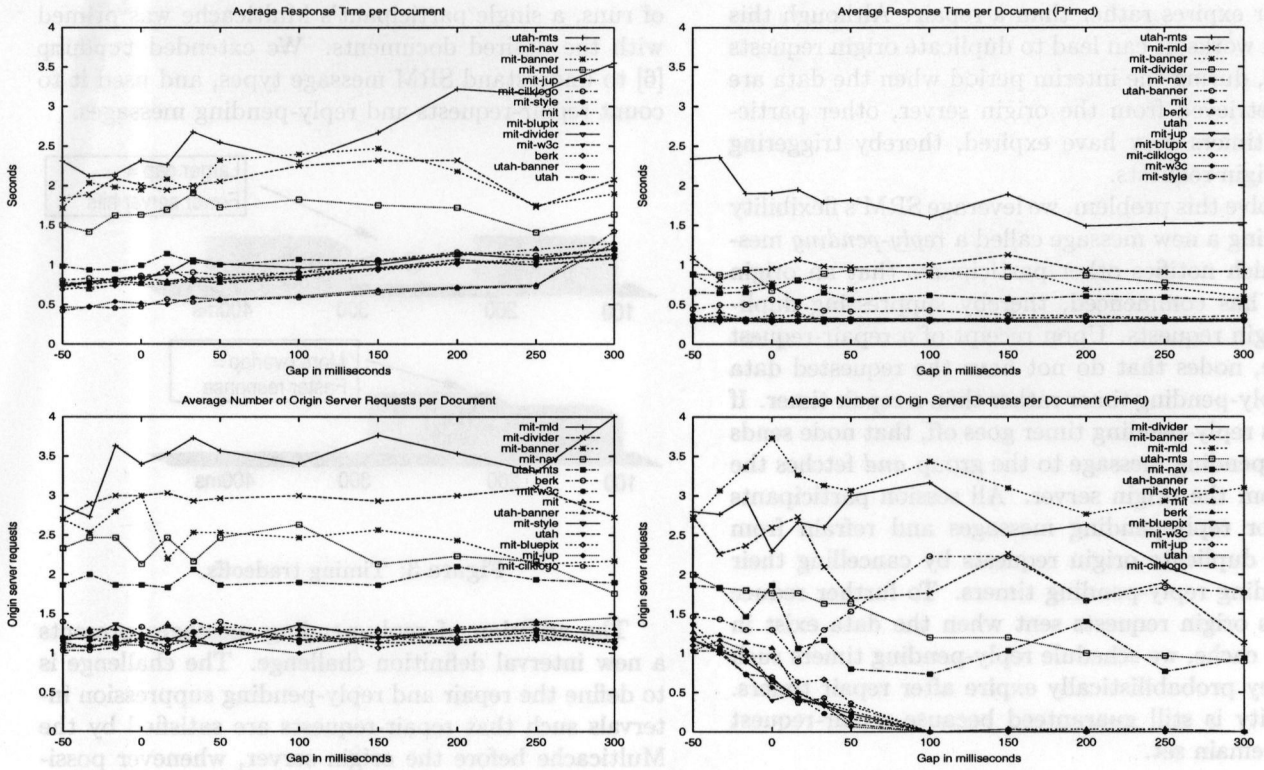

Figure 4: An initially empty Multicache is shown on the left; the case where a single participant's Multicache has been primed is shown on the right. Most documents' response times behave as expected but exceptions include very large documents and documents whose origin server is distant. Note the disparate time scales for the X and Y axes.

tire document is received. Those repair-requests often triggered multiple origin requests.

The four documents `mit-divider`, `mit-nav`, `mit-mld`, and `mit-banner` also experience long response times caused not by size but by distance to their origin server. The large number of origin server requests for these documents shows that we would fare better by biasing the reply-pending interval with the distance to the origin server using a service such as [11] that passively determines round-trip times and other network characteristics.

5 Lessons learned using libsrm

In this section, we report on our experience using *libsrm*. *libsrm* is a new instantiation of the framework presented in [5]. In addition to the SRM reliability mechanisms, *libsrm* incorporates the SNAP hierarchical, scalable naming protocol [10].

In general, we found *libsrm* simple to use. All of the SRM extensions described in section 3.2 were implemented in approximately 1200 lines of Tcl code which allowed for easy policy modification requiring no recompilation. This ease of change facilitated the experiments discussed in section 4. Lazy recovery was implemented simply by adding a one-line check to see if the URL had been explicitly requested in the application's implementation of the *libsrm* upcall invoked when a loss is detected.

We found that, while we gained critical namespace scalability from SNAP our application didn't conform naturally to SNAP's naming convention. One assumption central to both SNAP and SRM is that data are source-specific. Accordingly, a source is associated with each data name. While this association is desirable for the vast majority of applications, it is not optimal for webcast. Associating an SRM source with Web documents can falsely differentiate the same document. For example, node A might obtain and source `www.cs.berkeley.edu`, naming it `[A,www.cs.berkeley.edu]`. Simultaneously, node B gets the same URL but names it `[B,www.cs.berkeley.edu]`. Fortunately, duplicates can be easily detected at the application level where, in accordance with ALF principles, caching and recovery are managed so that duplicate cache entries are avoided. However, duplicate transmissions do sometimes occur (see below). A non-source-specific naming option would be desirable for transmitting web data. After limited analysis, we believe this can be accomplished by having all participants use the same, predefined, generic source. This would eliminate the false differentiations while maintaining SNAP's namespace scalability.

Finally, we learned that the natural choice for a Web application data unit (ADU) is not optimal. At the outset, the single document or image corresponding to a URL seemed natural. However, this proved to be too coarse-grained for larger documents. Since the SRM layer doesn't notify the application layer that an ADU has arrived until after the entire ADU is received, we occasionally observed the application, unaware that a document was already partially received and would shortly be fully received, spuriously request the document a second time. Compounded with the naming mismatch described above, this led to a transient state in which different SRM names existed for duplicate URLs and duplicate data were multicast. Depending on the document's size, the transmission of duplicates could account for significant amounts of unnecessary data. In hindsight, we would break documents into large fragments so that the application would be notified in a more timely manner of incoming documents. We implemented an earlier version of MASHCast designed for a non-fragmenting SRM library that *did* perform fragmentation and reordering. We should be able to merge this code into the current version with little change other than to expand the fragmentation granularity from 1K to 10K. This size avoids duplicating SRM's underlying 1K fragmentation efforts and avoids fragmenting in the common case of small documents.

6 Conclusions

We have described a novel webcast design that features reusable components and a custom protocol built upon the SRM framework. We have implemented our design in a useful collaborative application that is freely available for download. Through simple experimentation, we have verified the efficacy of our design and reported on our experience using *libsrm*.

Acknowledgements

We thank Suchitra Raman, Tina Wong and Yatin Chawathe for the implementation of *libsrm* and significantly improving MASHCast, and Yatin for conceiving the URL mangling scheme.

References

[1] E. Burns et al. Webcast – Collaborative Document Sharing via the MBone. Information online at http://www.ncsa.uiuc.edu/.

[2] D. Clark and D. Tennenhouse. Architectural considerations for a new generation of protocols. In *ACM SIGCOMM '90*, September 1990.

[3] G. Dauphin. mMosaic: Yet Another Tool Bringing Multicast to the Web. In *Workshop on Real Time Multimedia and the WWW*, October 1996.

[4] S. Deering and D. Cheriton. Multicast Routing in Datagram Internetworks and Extended LANs. *ACM Transactions on Computer Systems*, 8(2):85–110, May 1990.

[5] S. Floyd, V. Jacobson, S. McCanne, C.-G. Liu, and L. Zhang. A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing. In *Proceedings of the 1995 ACM SIGCOMM Conference*, August 1995.

[6] V. Jacobson, C. Leres, and S. McCanne. The Tcpdump Manual Page. Lawrence Berkeley Laboratory, Berkeley, CA, June 1989.

[7] T. Liao. Webcanal: a Multicast Web Application. In *6th International WWW Conference*, Santa Clara, CA, April 1997.

[8] S. McCanne et al. Toward a Common Infrastructure for Multimedia-Networking Middleware. In *Workshop on Network and OS Support for Audio and Video '97*, 1997.

[9] P. Parnes et al. mWeb: a framework for distributed presentations using the WWW and the MBone. In *W3C Workshop: Real Time Multimedia and the Web*, Sophia Antipolis, France, October 1996.

[10] S. Raman and S. McCanne. Scalable data naming for application level framing in reliable multicast. In *ACM Multimedia '98*, Bristol, UK, September 1998.

[11] S. Seshan, M. Stemm, and R. H. Katz. SPAND: Shared Passive Network Performance Discovery. In *Usenix Symplosium on Internet Technologies and Systems*, Monterey, December 1997.

[12] K. Wright. Using *libsrm* for Caching and Collaboration. Technical report, University of Utah, 1998. ftp://-mancos.cs.utah.edu/papers/libsrmcc.ps.gz.